Baptiste Provendier, bpp2218, 01553706

1. Network Training

To start with, we are going to examine the effect of data augmentation on accuracy. We define two data augmentation strategies. The first one will only apply a horizontal flip operation. The second one, much more aggressive, will combine a vertical flip with random zooming, translation and rotation of the input images. The loss curve of the original model quickly overfits due to the lack of data. This problem is vastly reduced with the light augmentation model with a final loss of 0.85. With the aggressive augmentation model, we do not overfit at all anymore but we get a much smaller accuracy overall. If used with precaution, data augmentation can help increase the accuracy and reduce overfitting by adding variance to the data. However, by applying an over aggressive method, we harm the performance of the test set by inputting data too far off the images in the dataset.

Model Tuning Parameter	**Best Validation Accuracy (%)**
No Data Augmentation	79.68
Light Data Augmentation	81.87
Aggressive Data Augmentation	59.36
Dropout	82.85
Batch Normalisation	77.85
Batch Normalisation + Dropout	84.46
Kernel Initialization	10.00

Table 1: Best validation accuracy for different tuning parameters

Next, we examine the impact of adding dropout and batch normalization. We are going to add a dropout layers of rate 0.3 after the last three three convolutional layers. Dropout ables us to keep the validation loss closer to the training loss by randomly deactivating some of the neurons in the model and hence preventing co-adaptation of neurons and overfitting [1]. We are able to learn better and we obtain a better accuracy. Batch normalization is a tool to center and rescasle the features of the model which helps diminish the reliance of gradients on the scale of the parameters [2]. It is particularly useful for activation functions that suffer from saturation (such as tanh and sigmoid) but it has very little impact on the ReLu activation. On its own, we do not see an improvement but coupled with dropout, we observe a better validation accuracy (batch normalization was applied on for the last 3 layers).

We now set the kernel initializer to 'zeros'. By doing that, the derivatives with respect to the loss function are all the same throughout the layers and the weights can't update and the model cannot learn. We get a constant validation accuracy of 10% which is equivalent to a linear mode, or a random guess on CIFAR-10 [3].

Lastly, we observe the loss curves depending on different learning rates. The learning rate in SGD corresponds to the step size when updating the weights. Choosing a smaller LR might be a good idea so that we won't miss a local minima but it results in increased training time. Increasing the LR will converge faster but risk to diverge from the global minimum. Even by using LR=3e-3, we do not get a plateau curve, which suggests we could have used a larger LR [4].

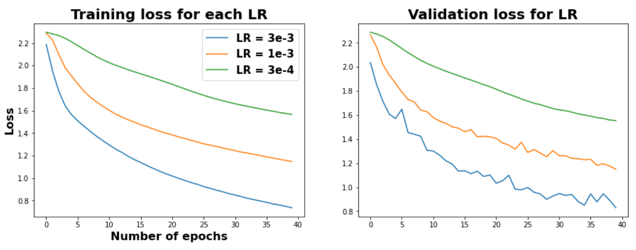


Figure 1: Training and Validation Loss curves depending on the SGD learning rate

2. Common CNN Architecture

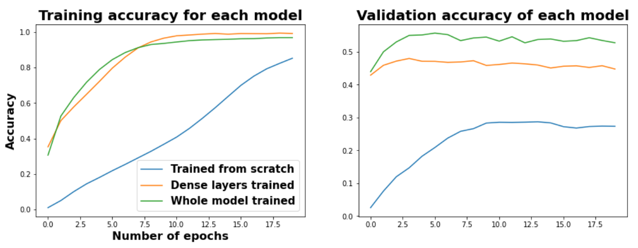


Figure 2: Training and validation accuracy curves for the VGG16 model trained from scratch, the transfer learning VGG16 model and the fine-tuning VGG16 model

Here, we observe the advantage of using transfer learning compared to training a model from scratch. The Vgg-16 architecture is designed for a much larger dataset and extracts features that are too complex for this task which results in poor generalisation. Starting with non-zero weights chosen from a simlar task, the pre-trained models achieved much better training accuracy very fast and significantly better validation accuracy as well. Transfer learning is a powerful tool to save up learning time as we start with a basis that we only need to build on. We also see that the model training only the dense layers is not as great as the one trained on the whole model. That is because by only training dense layers, the model is not specifically adapted to our task and cannot learn as great as by training all the layers since the weights are not updated.

Model name	Test acc.	Train time	Inf. time
Vgg scratch	26.87%	1,139s	0.31ms
Vgg Transfer Learning	44.3%	429s	0.31ms
Vgg Finetunning	52.4%	1,134s	0.31ms
Xception	58.1%	1,594s	0.32ms
NasNetMobile	65.5%	10,332s	0.88ms

Table 2: Evaluating Tiny Imagenet results from different models

The most notable difference will be that transfer learning is far more time efficient that the rest. Fine tuning will adapt the whole model to the task and hence will take more time to train but will give a better accuracy whereas transfer learning only changes the weights within the dense layers. The GPU used to record the inference times is a Tesla P100-PCIE-16GB. For the additional models, we first used Xception which took around the same training time as Vgg16 but achieved a better accuracy and NASNetMobile, which is much longer to run (around 9 minutes per epoch) but achieved far better results to classify Tiny-ImageNet. Both models had better Top-1 accuracy with fewer parameters than Vgg16 (see appendix) and the weights were initialized with imagenet. For both models, a compromise has to be done between better test accuracy and longer training times.

3. RNN

RNN Regression In this task ,we observe the effect of the window size, which is the number of past observations used to predict the next value, on the test prediction curves. The best results are obtained with a window size of 12, which makes sense since we are dealing with periodic data over year time. Using the data for the past year to predict the number of passengers for the next month hence gives you the most information. We obtain a much better MSE (mse = 34) compared to using a window size of 1 (mse = 52) or 24 (mse = 95). Many more window size values were tested to define the best MSE score but only these three were chosen as graphical representation to keep the curves readable.

Text Embeddings This task tests the effect of transfer learning in the emdeddings by comparing 3 models trained

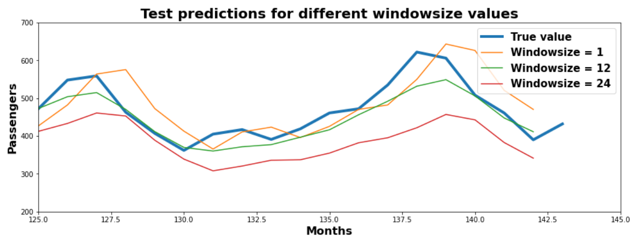


Figure 3: Test curves for different models for different window size values

on the IMdB sentiment dataset. The first model has embedding of dimension 1 and average pooling, the second is an LSTM initialized with trainable embeddings initialised at random and the last model is an LSTM with untrainable Glove embeddings. Looking at the accuracy curves we observe the same training time advantages of transfer learning as previously seen. The LSTM models quickly get their maximum accuracies and even overfit whereas the embedding model continues to learn. Surprisingly, even with no LSTM unit and an embedding of dimension 1 the Embeddings model achieves similar validation accuracy compared to the two other models. It can be explained, in part, with the simplicity of the task. The Embeddings model will classify any review containing negative word as negative sentiment. Generally, the snetiment of individual words will be a good indicator of the overall sentiment of a sentence, hence achieving high test accuracy. However, testing the model on specific sentences, we can ”trick” it and show its weaknesses. The Embeddings model will classify both the negative and positive reviews with negative sentiment (value to 0) as the phrases contains negative words and the model does not the capacity to capture the structure of the sentence. With a higher embedding dimension, the Glove model is able to detect the positive sentiment with its bigger capacity and ability to capture the sequential context; whereas the Embeddings model acts as a one to one RNN.

Model name	Test accuracy	Sentiments
Embeddings	85.22%	6e-6 —— 6e-6
LSTM	85.08%	N/A
LSTM glove	86.04%	0.01 —— 0.69

Table 3: Evaluating Tiny Imagenet results from different models

Another difference are the closest words to 8 proposed by the embeddings. By training the Embeddings model on the IMdB dataset, it encodes the property of the words in the movie reviews context and is able to associate ’8’ with words that characterise a positive review. On the other hand, by initializing the LSTM model with Glove, the model encodes words in a more general sense, not specific to movie reviews at all. This is why the most similar words it can found are the closest numbers to ’8’ (see appendix).

Text Generation Finally, we observe the effect of temperature on the BLEU score for models on different levels. In the general sense, a temperature closer to 0 will output safer predictions whereas larger temperatures will follow the probability distribution of the model output and increases text variability. 10 different temperatures were used to evaluate the BLEU score for each model. The BLEU score was designed to evaluate text translation and looks for matches on a word level between the generated text and the reference text. For temperatures close to 0, both models behave in a similar way; they repeat a lot of words and even sentences as the variability is very low. In both cases, the models receive good BLEU scores which demonstrates that the score evaluates the correctness of the text rather than its diversity and whether the sentences make sense or not. In the case of the character level model, as the temperature increases, it will decide on the next character based on the probability distribution of the original output more and more rather than taking the safest option with regard to the previous character. That means that for a temperature of 2, the model outputs words that are grammatically wrong with characters other than letters which leads to a very low BLEU score. The same phenomena can be observed with the word-level model, however since the model makes predictions word by word following the distribution from the script, it keeps relevance and similarities with the training data, hence keeping a high BLEU score. Observations can be made that the BLEU score is more effective as a tool to measure correctness of a text and more specifically on a character level basis.

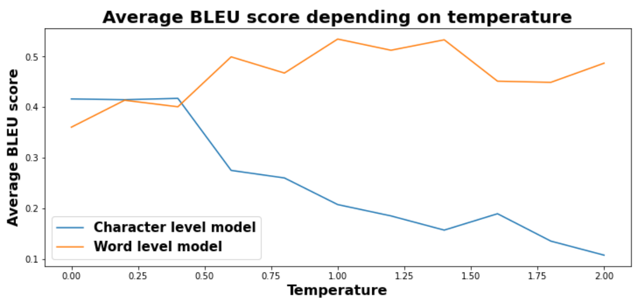


Figure 4: BLEU score values depending on temperature

4. Autoencoders

Non-Linear Transformations for Representation Learning In this section we use three methods to try to reduce the representation of the Cifar-10 dataset to 10 latent dimensions. The three models are: Principal Component Analysis (PCA), Linear Autoencoder (LAE) and Convolutional Autoencoder (CAE). The architecture of the LAE is: **Dense(2048), D(1024), Rep(10), D(1024), D(2048), D(1024), Reshape**.The architecture for CAE is: **Conv(128), MaxPooling, Conv(256), Rep(10), D(1024), D(2048), D(1024), Reshape**. All layers used ReLu activation with the convolutional layers having (3x3) filters, stride of 1 and same padding. Both the linear AE and the convolutional AE show significantly better performance compared to PCA. The PCA model is restricted to linear mapping and transformation while autoencoders can model non-linear projections [5] allowing for higher dimensionnal data. The fact that the image of Cifar-10 dataset are limited in their spatial information gives little advantage to the features extracted in Convolutional AE compared to the linear AE and even slightly hurts the accuracies and MSE.

Model	MSE		Accuracy (%)	
Name	train	val	train	val
Linear autoencoder	0.006	0.008	93.60	93.45
Conv. autoencoder	0.013	0.015	92.60	92.90
PCA	0.026	0.025	81.20	81.42

Table 4: Evaluating MSE and accuracy of different methods on Cifar-10 dataset

Custom Loss Functions In this task we aim to denoise images from the Cifar-100 dataset and observe the MSE from a baseline UNet model trained using different loss functions. We use a classic MSE loss function, a Structural Similarity Index (SSIM), a Mean Absolute Error (MAE) and a Peak to noise Ratio (1/PSNR). As the aim is normally to maximise SSIM, we use 1-SSIM as a loss to adapt it to a minimisation problem [6].

Loss Function	Test MSE
MSE	0.0072
SSIM	0.0085
MAE	0.0056
1/PSNR	0.0054

Table 5: Test MSE for different loss functions

Numerically, looking at the MSE, the PNSR and MAE loss functions obtain better scores. Looking visually at the pictures, we see that the PSNR outputs sharper images than the other loss functions. This can be explained by how the PSNR is calculated and the fact that it is much more sensitive to added noise than ohter loss functions like SSIM [7]. In fact, it is a metric to use carefully as it can have a biased toward over smoothed images (ones that not only remove the noise but also part of the texture of the image) [8].

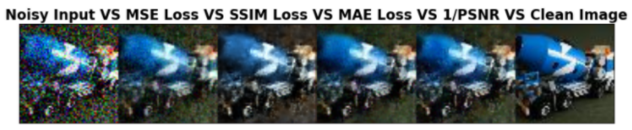

Noisy Input VS MSE Loss VS SSIM Loss VS MAE Loss VS 1/PSNR VS Clean Image

Figure 5: Denoised images using different loss functions

5. VAEs and GANs

MNIST Generation using VAE and GAN In this section we train three models, variational autoencoders (VAE) with and without the KL divergence and a Generative Adversarial Network (GAN), on the MNIST dataset and compare their performances based on the resulting Mean Squared Error (MSE) and Inception Score (IS) score.

Model name	MSE	Inception score (IS)
VAE w/ KL	0.0117	7.20
VAE w/o KL	0.0109	5.84
GAN model	N/A	8.28

Table 6: MSE and IS for different models of VAE and GAN

The Inception score is a quantitative way to evaluate the quality of synthetically created images [9]. The score seeks to capture two properties from the generated images which are their quality, can they easily be classified in the right category, and their diversity, can a wide range of object be generated [9]. The KL divergence loss is a regularisation term that stops the model from overfitting by minimising the reconstruction error. It encourages the encoder to distribute all the inputs and penalizes it from clustering data into regions [10]. KL divergence will try to 'center' the data to get continuity between the latent and output space and erase the 'empty' regions to get output that can be distinctly classified [11]. This regularisation will inevitably increase the MSE (reconstruction error) but will also improve the inception score as observed in the table. However, looking at the Inception Score from the table we see that GANs outperforms all forms of VAEs previously discussed. This is because GANs are explicitly set up to optimise for generative tasks [12] and can hence produce more varied, and still distinct, data compared to VAEs.

Quantitative VS Qualitative Results In this task, we try to colour black and white images using a cGAN approach predicting pixel wise values of the image compared to a UNet autoencoder trained with mean absolute error (MAE). We include in the appendix the predictions for three sets of images and compare them to the real pictures. The MAE model is trained to minimise the reconstruction error, and thus tends to make 'safe' yet unrealistic predictions. Quantitatively, it has a lower MAE than the cGAN trained model but results are poor. The cGAN approach uses a min max loss function between the generator and discriminator to differentiate between the 'real' and 'fake' data and the cGAN model thus trains itself to generate data that imitates the real data. As we can see in the images, using MAE loss alone, the model learns to colorize images but it will be **conservative** and use colors like 'gray' and 'brown' when it doubts which color to use and will just try to reduce the MAE loss as much as it can [13]. This technique give more realistic results like in the first picture where the real picture is a blue truck and the cGAN outputs a red truck, which will receive a low reconstruction score but looks better, or in the second picture where cGAN outputs blue water instead of green like the real picture even though it looks better qualitatively.

6. Deep Reinforcement Learning

In this task, we explore different techniques and policies on Deep Reinforcement learning. We observe the average reward for the past 50 episodes for 4 different strategies tested on the CartPole problem. First, we use Q-Learning with an ϵ greedy policy, then we use that same policy with SARSA. The last two techniques use the softmax policy on Q-Learning and SARSA.

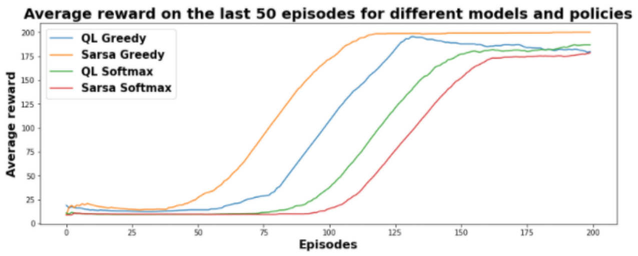


Figure 6: Average reward for the last 50 episodes for different models and policies

The code modifications to implement the different agents are detailed in the appendix. The main difference between Q-Learning and SARSA is the fact that SARSA in an on-policy method while Q-Learning is off-policy. The first thing we notice is that off-policies methods have higher per-sample variance on-policy learning, hence, when using ϵ-greedy policy, the SARSA model directly improves exploration on that specific policy and it converges faster [14]. On the other hand, Q-Learning learns an optimal policy whilst exploring a new one. This approach offers robust learning but there is a high change that the agent chooses a random subobtimal action as the next step which can lead to longer convergence time and more errors [15]. For both models, using a softmax policy reduces the performance as we removed the ϵ term which directly decreased the degree of convergence over time, we now aim to learn the degree of exploration by sampling actions from a probability distribution according to the performance of actions in the state-action space. Using that policy will further reduce convergence and decrease the average score [16]. It is worth noting that the results for each model vary from experience to experience due to high dependency on the initial state of the algorithms.

References

[1] https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5

[2] https://towardsdatascience.com/batch-normalisation-in-deep-neural-network-ce65dd9e8dbf

[3] Lecture Notes: Part 3.2 Slide 18

[4] https://towardsdatascience.com/understanding-learning-rates-and-how-it-improves-performance-in-deep-learning-d0d4059c1c10: :text=Furthermore

[5] https://towardsdatascience.com/autoencoders-vs-pca-when-to-use-which-73de063f5d7: :text=PCA

[6] https://stackoverflow.com/questions/57357146/use-ssim-loss-function-with-keras

[7] https://ieeexplore.ieee.org/abstract/document/5596999

[8] https://stackoverflow.com/questions/10465003/image-processing-does-psnr-and-ssim-metrics-show-smoothing-noise-reduction-q

[9] https://machinelearningmastery.com/how-to-implement-the-inception-score-from-scratch-for-evaluating-generated-images/: :text=The

[10] https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf

[11] https://www.reddit.com/r/MachineLearning/comments /46xjtw/how_to_understand_the_kl_divergence_term_in/

[12] https://www.reddit.com/r/MachineLearning/comments/ 4r3pjy/variational_autoencoders_vae_vs_generative/

[13] https://towardsdatascience.com/colorizing-black-white-images-with-u-net-and-conditional-gan-a-tutorial-81b2df111cd8

[14] https://stats.stackexchange.com/questions/326788/when-to-choose-sarsa-vs-q-learning

[15] https://subscription.packtpub.com/book/data/9781789345 803/1/ch01lvl1sec13/sarsa-versus-q-learning-on-policy-or-off

[16] https://medium.com/analytics-vidhya/multi-armed-bandit-analysis-of-softmax-algorithm-e1fa4cb0c422

7. Appendix

7.1. Appendix for section 1 :

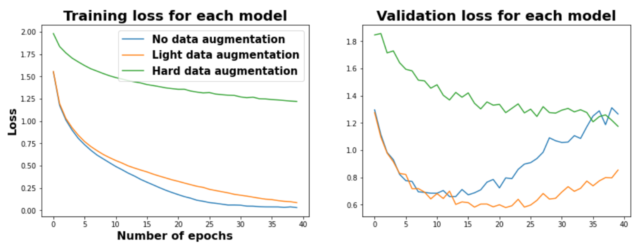


Figure 7: Lab 3: Training and validation curves for the original model and the two data augmentation strategies

7.2. Appendix for section 2 :

Model name	Top-1 Accuracy	Number of parameters
VGG16	71.3%	138 million
XCeption	79.0%	22 million
NASNetMobile	74.4%	5 million

Table 7: Comparaison of Top-1 accuracy and number of parameters for the different models used

7.3. Appendix for section 3 :

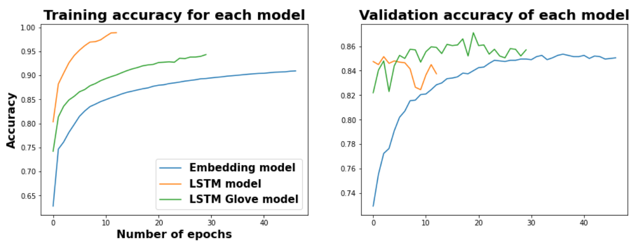


Figure 8: Lab 5: Training and validation curves for the embedding model, the LSTM model and the LSTM glove model

Similar words to 8	Embeddings model	LSTM Glove
1	excellent	9
2	highly	7
3	wonderful	6
4	perfect	5
5	recommended	4
6	superb	12
7	9	3
8	7	10
9	favorite	16
10	today	13

Table 8: Top10 most similar word to 8 for Emdeddings and LSTM Glove models

7.5. Appendix for section 5 :

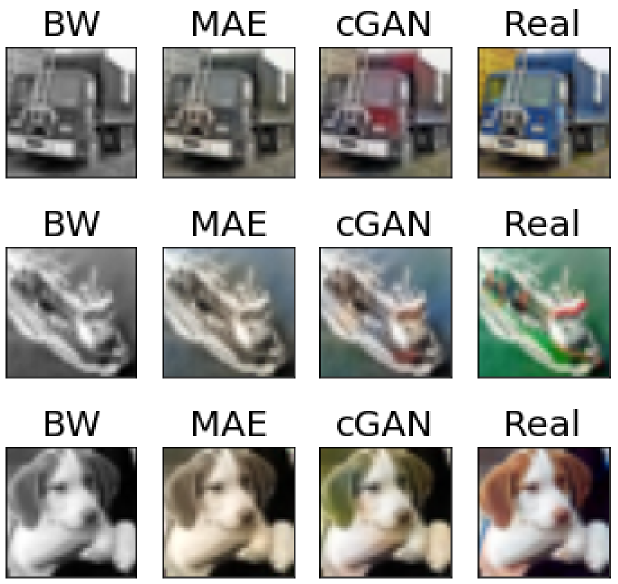
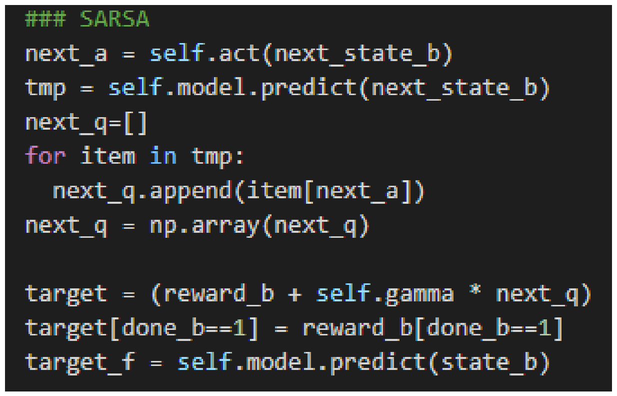


Figure 9: Comparing BW pictures with coloured versions using a MAE and a cGAN with the real picture

7.6. Appendix for Section 6 : Changes made to the code to implement the SARSA model. We need to change the replay part of DQNAgent. Instead of just taking the max value of Q, we need to find the next value of Q and put in the target function. Following the equation from the lecture notes, we first need to find the (a+1) term using the next state. We then create a term containing all the possible predictions for the following state. We obtain an array of arrays containing all the actions depending on the next state. We go through the array and select the actions q associated with our next state and put that term in the target function instead of the maximum value that was there before for q-learning.

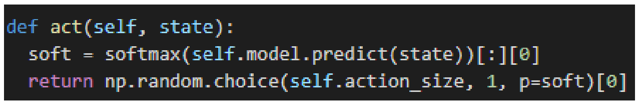


```
### SARSA
next_a = self.act(next_state_b)
tmp = self.model.predict(next_state_b)
next_q=[]
for item in tmp:
  next_q.append(item[next_a])
next_q = np.array(next_q)

target = (reward_b + self.gamma * next_q)
target[done_b==1] = reward_b[done_b==1]
target_f = self.model.predict(state_b)
```

Figure 10: Changes made to the replay part of DQNAgent of the code to implement the SARSA model

Changes made to the code to implement the softmax policy. We need to change the act part of DNQAgent. To do that we first get the prediction of the next actions depending of the current state and apply the softmax function to it. That function will output softmax probabilities for each sets of possible actions. We then use the random choice numpy function. We choose a random action from the list of possible actions taking into input the list of probabilities outputed with softmax.



```
def act(self, state):
  soft = softmax(self.model.predict(state))[:][0]
  return np.random.choice(self.action_size, 1, p=soft)[0]
```

Figure 11: Changes made to the replay part of DQNAgent of the code to implement the softmax policy